Microsoft SQL Server 7.0 Security

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

© 1999 Microsoft Corporation. All rights reserved.

Microsoft, BackOffice, Visual Basic, Win32, Windows, and Windows NTare either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Other trademarks and tradenames mentioned herein are the property of their respective owners.

Introduction **1**

Contents

Assumptions **2**

New Security Features of SQL Server 7.0 ..... **3**
- Security Modes 3
  - Windows NT Authentication Mode 3
  - Mixed Mode 3
- Using SIDs Internally 4
- Roles 4
  - Public Role 4
  - Predefined Roles 4
    - Fixed Server Roles 4
    - Fixed Database Roles 5
  - User-Defined Roles 5
- Application Roles 6
- Permissions 6
  - Permissions System 7
  - Deny Permission 7
- Delegation 7
- Networking Protocols 8

Implementing the SQL Server 7.0 Security Model **9**
- Securing Access to the Server 11
  - Windows NT Authentication Mode 11
    - Windows NT Level 11
    - Server Level 12
  - Mixed Mode 13
    - Server Level 13
  - Fixed Server Roles 14
- Securing Access to the Database 14
- Securing Access to the Database Objects 15
  - User-Defined Database Roles 15
  - Granting and Denying Permissions to Users and Roles 17
  - Ownership Chains 19
- Securing Access Through Application Roles 20

Security Internals **23**
- Implementation of Server Level Security 23
  - Use of SIDs 23
  - Generation of GUIDs for Non-Trusted Users 23
  - Renaming Windows NT User or Group Accounts 23
  - sysxlogins System Table 24
    - xstatus Column 24
    - dbid and language Columns 25
    - hasaccess Status 26
    - denylogin Status 26
  - sysremotelogins View 26
  - sysoledbuser View 26

Implementation of Object-Level Security....27
How Permissions Are Checked....27
Cost of Changing Permissions....28
Renaming Windows NT User or Group Accounts....28
sysprocedures System Table Removed....29
With Grant Option....29
sysusers System Table....30
hasdbaccess column....30
sysmembers System Table....30
syspermissions System Table....30
sysprotects System Table....31
Named Pipes and Multiprotocol Permissions....31

Upgrading from SQL Server 6.5....33
Upgrade Considerations....33
Upgrade Process....33
Analyzing the Upgrade Output....33
User Has Been Deleted....34
Administrator Account....34
Users of a Trusted Domain....34
Users of the Current Default Domain....35
How to Reduce the Upgrade Risk....35
Preparing SQL Server 6.5 Security Environment....35
Step Through the Upgrade....35
Do Not Upgrade to a New Domain....35
Character Mapping Not Required....35
Do Not Use sa Account....36
Do Not Use Aliasing....36

Setting Up a Secure SQL Server 7.0 Installation....37
Do Not Use the sa Account....37
Service Accounts....38
Local System Account....38
Local User Account....39
Domain User Account....39
File System....39
Registry....40
Auditing....40
Profiling for Auditing....40
Backup and Restore....41
Security of Backup Files and Media....41
Restoring to Another Server....41
Mixed Mode....41
Windows NT Authentication (Same Domain)....42
Windows NT Authentication (Different Domain)....42
Users from a Trusted Domain....42
Users from the New Domain....42
Users from Any Domain with Same Username and Password....43
Attaching/Detaching Database Files....43
General Windows NT Security Configurations....43
Disable Windows NT Guest Account....43
Restrict Physical Access....43

Conclusion....44

Appendix A: Finding More Information.....45
Books.....45
Web Sites.....45
White Papers.....45

Appendix B: Security Related Stored Procedures.....46

Appendix C: Changes to SQL-DMO Object Model.....47
Removed SQL-DMO Objects/Collections.....47
Changed SQL-DMO Objects/Collections.....47
New SQL-DMO Objects/Collections.....48

Introduction

With the arrival of Microsoft® SQL Server™ version 7.0, Microsoft's offering in the database arena has improved significantly. The enhancements to this product are by far the most significant of any prior SQL Server release. There have been major enhancements in the areas of scalability, robustness, features, functionality, and security. For the first time since Microsoft released SQL Server version 1.0 in 1989, Microsoft Windows NT® security has been completely integrated into the core of the product.

The first integration of Windows NT security with SQL Server occurred with the release of SQL Server version 4.2, when integrated mode was introduced. Further enhancements were released with SQL Server version 6.0, as Microsoft tried to provide the most practical security framework within the confines of the existing legacy code.

Many developers and administrators successfully implemented the security architecture of earlier versions of SQL Server. However, those who took security seriously were certainly aware that the improvements to security could be classed only as patches until a more thorough integration was possible. This integration could be made possible only through the redevelopment of the SQL Server core features. SQL Server 7.0 provided that opportunity.

The purpose of this document is to introduce SQL Server administrators and developers to the new security features of SQL Server version 7.0. Outlining the new features and providing a detailed discussion on how to best implement security in a Windows NT domain environment achieves this purpose. Source code examples are included for the benefit of developers who want to implement the new security model immediately.

For those who will be upgrading servers from earlier versions, information that explains the security part of the upgrade is provided. Tips are also provided for those who are knowledgeable in the way security was performed in versions earlier than SQL Server 7.0.

With the SQL Server 7.0 release, SQL Server security seamlessly integrates with the Microsoft BackOffice® product suite.

Assumptions

Microsoft SQL Server 7.0 security is based on the Windows NT security model; therefore, a fair understanding of Windows NT security is assumed throughout this document. The reader is expected to understand the concept of domains, global groups, local groups, and user accounts as they apply in the context of Windows NT security.

An understanding of SQL Server 6.x security, while advantageous, is not required for most of the document. However, an exception to this is "Upgrading from SQL Server 6.5." This section specifically addresses how

security issues in SQL Server version 6.5 are addressed in SQL Server version 7.0.

Those who were familiar with the system tables of SQL Server 6.5 and earlier will have an advantage in comprehending the various topics in “Security Internals.”

For those interested in the code examples, exposure to Microsoft Visual Basic® and Structured Query Language (SQL) will greatly enhance understanding. Prior experience using SQL Distributed Management Objects (SQL-DMO) provides a significant advantage as well.

However, it is hoped that even those who have had limited exposure to Windows NT or SQL Server will gain a significant amount of understanding about the workings of security when using these products. This is particularly true of the first two sections: “New Security Features of SQL Server 7.0” and “Implementing the SQL Server 7.0 Security Model.”

New Security Features of SQL Server 7.0

Security Modes

Microsoft SQL Server 7.0 provides two modes for securing access to the server: Windows NT Authentication Mode and Mixed Mode.

In Windows NT Authentication Mode, Microsoft SQL Server 7.0 relies solely on the Windows NT authentication of the user. Windows NT users or groups are then granted access to the SQL Server. In Mixed Mode, users may be authenticated by Windows NT or by SQL Server. Users that are authenticated by SQL Server have their username and password pairs maintained within the SQL Server.[1]

The standard mode of SQL Server 6.5 and earlier has been discontinued. The practical relevance of this is that SQL Server 7.0 will always allow Windows NT-based logons to take place.

Windows NT Authentication Mode

This security mode allows SQL Server 7.0 to rely on Windows NT to authenticate users in much the same way as other applications. Connections made to the server using this mode are known as trusted connections.

---

[1] The username and password pairs are stored in the **sysxlogins** system table of the master database.

When Windows NT Authentication Mode is used, the database administrator allows users to access the computer running SQL Server by granting them the right to log in to SQL Server 7.0. Windows NT security identifiers (SIDs) are used to track Windows NT authenticated logons. As Windows NT SIDs are used, the database administrator can grant logon access directly to Windows NT users or groups.

Mixed Mode

In SQL Server 7.0, Mixed Mode relies on Windows NT to authenticate users when the client and server are capable of using NTLM[2], or Kerberos logon authentication protocols. If either party is incapable of using a standard Windows NT logon, SQL Server requires a username and password pair, and compares this pair against those stored in its system tables. Connections that rely on username and password pairs are called non-trusted.

Mixed mode is supplied for two reasons: backward compatibility and when SQL Server 7.0 is installed on the Microsoft Windows® 95 and Windows 98 operating systems, as trusted connections are not supported on Windows 95/98 computers when they are the “Server.”

Using SIDs Internally

One of the major enhancements of SQL Server 7.0 is that it now uses security identification numbers (SIDs) internally. Windows NT users and groups can be granted access to databases or specific database objects directly. For example, Jane is a member of the **SALES** and **MARKETING** groups in Windows NT. The **SALES** group has been granted permission to log in to SQL Server, and also to access the **pubs** database. An administrator could grant access to the **authors** table for Jane by her Windows NT name, REDMOND\Jane. The Windows NT account must be referenced by domain and username. In this case, Jane’s SID would be stored in the system tables of the **pubs** database.

Roles

Roles allow users to be collected into a single unit against which permissions can be applied. Permissions granted, denied, or revoked from a role also apply to any members of the role. Roles can represent a job performed by a class of workers in an organization. Permissions can then be granted to that role. As workers rotate into the job, they are made members of the role; as they rotate out of the job, they are removed. This removes the requirement to repeatedly grant, deny, and revoke permissions to or from individuals as they accept or leave a job. Roles are used much like Windows NT groups are used.

There are a number of key concepts that make roles so powerful. First, with the exception of fixed server roles, they are implemented within a

[2] Standard Windows NT logon using challenge-response.

database. This means that the database administrator is not reliant on the Windows NT administrator for the grouping of users. Second, roles can be nested. This nesting is not limited by levels, but for obvious reasons does not allow circular nesting. Third, unlike groups in SQL Server 6.5 and earlier, a database user can be a member of more than one role simultaneously.

Public Role

The public role is a special role that exists in every database, including the system databases **master**, **msdb**, **tempdb** and **model**. Functionally, it can be compared to the **Everyone** group in the Windows NT environment. Every database user is a member of this role automatically; hence users cannot be added or removed from this role. This role provides the default permissions for users in a database and cannot be deleted.

Predefined Roles

SQL Server 7.0 ships with several predefined roles. These roles have predefined implied permissions, which cannot be granted to other user accounts.

There are two types of predefined roles: fixed server roles and fixed database roles.

Fixed Server Roles

Fixed server roles are server-wide in their scope. They exist outside of the databases. Each member of a fixed server role is able to add other logins to that same role.

Note: All members of the Windows **NT BUILTIN\Administrators** group (the local administrator's group) are members of the **sysadmin** role by default.

The following table lists the fixed server roles found in SQL Server 7.0.

Fixed server role	Description
sysadmin	Performs any activity in SQL Server.
serveradmin	Configures server-wide configuration options, shuts down the server.
setupadmin	Manages linked servers and startup procedures.
securityadmin	Manages logins and CREATE DATABASE permissions, reads error logs.
processadmin	Manages processes running in SQL Server.
dbcreator	Creates, alters, and restores databases.
diskadmin	Manages disk files.

For more information about the use of these roles, see SQL Server Books Online.

Note: The **dbcreator** role has the ability to restore databases.

Fixed Database Roles

Fixed database roles are defined at the database level and exist in each database. Members of the **db_owner** and **db_security** admin roles can manage fixed database role membership. However, only the **db_owner** can add others to the **db_owner** fixed database role.

The following table contains a list of the fixed database roles that ship with SQL Server 7.0.

Fixed database role	Description
db_owner	Performs all maintenance and configuration activities in the database.
db_accessadmin	Adds or removes access for Windows NT users, groups, and SQL Server logins.
db_datareader	Reads all data from all user tables.
db_datawriter	Adds, deletes, or changes data in all user tables.
db_ddladmin	Runs any Data Definition Language (DDL) command in a database.
db_securityadmin	Modifies role membership and manages permissions.
db_backupoperator	Backs up the database.
db_denydatareader	Cannot read any data in user tables within a database.
db_denydatawriter	Cannot add, modify, or delete data in any user tables or views.

For more information about the use of these roles, see SQL Server Books Online.

User-Defined Roles

User-defined roles provide an easy mechanism for managing permission in a database when a group of users performs a specified set of activities in SQL Server 7.0 and there is no applicable Microsoft Windows NT group, or if the database administrator does not have permissions to manage the Windows NT user accounts. In these situations, user defined-roles provide the database administrator the same flexibility as Windows NT groups.

User-defined roles apply only at the database level, and are local to the database in which they were created.

Application Roles

SQL Server 7.0 application roles allow the database administrator to restrict user access to data based on the application that the user is using. Application roles allow the application to take over the responsibility of user authentication.

When an application makes a connection to SQL Server 7.0, it executes the **sp_setapprole** stored procedure, which takes two parameters: username and password (which can be encrypted[3]). [4] The existing permissions assigned to the user are dropped, and the security context of the application role is assumed.

After application roles are activated, they cannot be deactivated. The only way to return to the original security context of the user is to disconnect and reconnect to SQL Server.

Application roles work with both authentication methods, and contain no members. Users cannot be associated with application roles, as the application requests the application role's security context using the **sp_setapprole** stored procedure.

Like user-defined roles, application roles exist within a database only. If, while an application is in the security context of an application role, another database is accessed, the access to the other database is, by virtue of permissions, granted to the **guest** account in that database. If the **guest** account has not been specifically granted access to the data, or does not exist, the objects cannot be accessed.

Another key concept in the use of application roles is that the user who is running the application is audited within SQL Server 7.0. In other words, application roles provide the security context within which the database object permissions are checked, but the identity of the actual user is not lost.

---

[3] The password can always be encrypted before being sent to SQL Server. If the Multiprotocol Net-Library is used, the packet containing the password can also be encrypted.

[4] The best way of storing the username and password in an application is probably to store the information in a registry key. The key should be encrypted, and only the application should have the key to decrypt it.

Here is an example of an implementation using application roles. If Jane is a member of the **ACCOUNTING** group, and the **ACCOUNTING** group members are permitted to gain access only to the data in SQL Server through the accounting software package, an application role could be created for the accounting software. The **ACCOUNTING** application role would be granted access to the data, while the **ACCOUNTING** Windows NT group would be denied access to the data. Thus, when Jane attempts to access the data using SQL Server Query Analyzer, she will be denied access; but when Jane uses the accounting software, she will be able to access the data.

Permissions

Due to the tight integration of the SQL Server 7.0 security model with Windows NT, it is recommended that users are grouped at the Windows NT level. Users and permissions as they exist within SQL Server 7.0 are discussed more in "Securing Access to the Server."

Permissions System

There has been a change in the way that permissions are implemented. In SQL Server 6.5, permissions were hierarchical. The **public** group was overridden by user-defined groups, which were overridden by permissions assigned to individual users. The hierarchy was clearly defined: user permissions took precedence over user-defined groups, and user-defined groups took precedence over the **public** group.[5]

In SQL Server 7.0, the permission system is based on the same additive model that forms the basis of Windows NT permissions. If a user is a member of the **sales**, **marketing**, and **research** roles (multiple group memberships are now possible), the user gets the sum of the respective permissions of each role. For example, if **sales** has SELECT permissions on a table, **marketing** has INSERT permissions, and **research** has UPDATE permissions, the user would be able to SELECT, INSERT, and UPDATE. However, as with Windows NT, if a particular role of which the user is a member has been denied a specific object permission (such as SELECT), the user is unable to exercise that permission. The most restrictive permission (DENY) takes precedence.

Deny Permission

This new permission in SQL Server 7.0 allows an administrator to deny an object or statement permission to a user or role. As with Windows NT permissions, the DENY takes precedence over all other permissions.

---

[5] In Microsoft SQL Server 6.0 and prior versions, the permission system was based on the principle of "Whatever permission is set last goes." For example, if the SELECT permission was granted to a user and then revoked from the group of which the user was a member, the user was not able to SELECT; this is because the last statement issued was to revoke SELECT from the group, of which the user was a member.

This can be very useful. For example, if one finds that some database users are frivolously changing data, it would not be fair to remove permissions from all users, as the majority of the users are using the data responsibly. It is possible to create a new role with a name like **trouble_makers**, and then DENY the INSERT, UPDATE, and DELETE operations on all tables for this role. As users misbehave, they are put into the **trouble_makers** role without regard for their other personal, group, or role permissions.

Delegation

Delegation is a permission that is documented in SQL Server Books Online, but is not in SQL Server 7.0. SQL Server 7.0 will accept delegated Kerberos tickets, but will not delegate these further (when running on the Microsoft Windows 2000 operating system). This affects remote stored procedures, which continue to operate as they did in SQL Server 6.5, and affects distributed queries, which do not support delegation in SQL Server 7.0.

Networking Protocols

SQL Server 7.0 is now able to use Windows NT Authentication Mode over all the network libraries. Prior to SQL Server 7.0, only the Named Pipes and Multiprotocol Net-Libraries were able to use trusted connections. This was often a problem as TCP/IP Sockets performed better over slow WAN links. Now, organizations with remote sites are able to have the performance of sockets with the functionality of Windows NT Authentication.

When installing SQL Server 7.0 on Windows NT, trusted connections are forced if the Named Pipes or Multiprotocol Net-Libraries are used. This is because Windows NT authenticates the user at the network level, before the connection is passed to SQL Server. If the TCP/IP Sockets Net-Library is used, SQL Server 7.0 establishes communication with the client and forces authentication through Windows NT prior to giving the user access to databases.

Note: For SQL Server 7.0 installed on the Windows 95 and Windows 98 operating systems, the default network library installed is the TCP/IP sockets Net-Library. This is because Windows 95 and Windows 98 do not

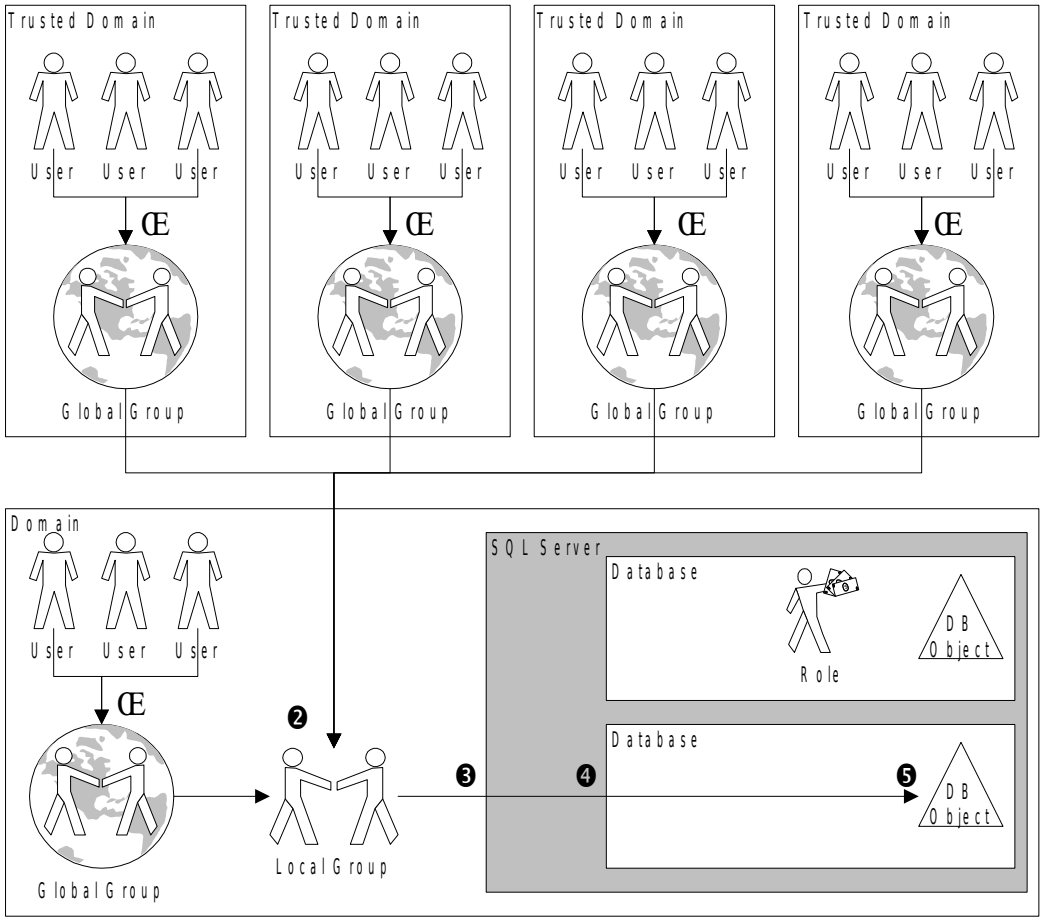
Implementing the SQL Server 7.0 Security Model

support server-side named pipes.[6]

In order to implement security in the most practical ways on SQL Server 7.0, it is important to understand how the design team expected the security model to be implemented. Those familiar with Windows NT security will notice the power that Windows NT users and groups bring to SQL Server 7.0.

Security on SQL Server 7.0 should be implemented as outlined in the following diagram.

[6] Windows 95 and Windows 98 operating systems can use named pipes as clients, but not as servers.



The steps in the diagram are summarized as follows:

1. Users in each domain are assigned to Windows NT global groups.

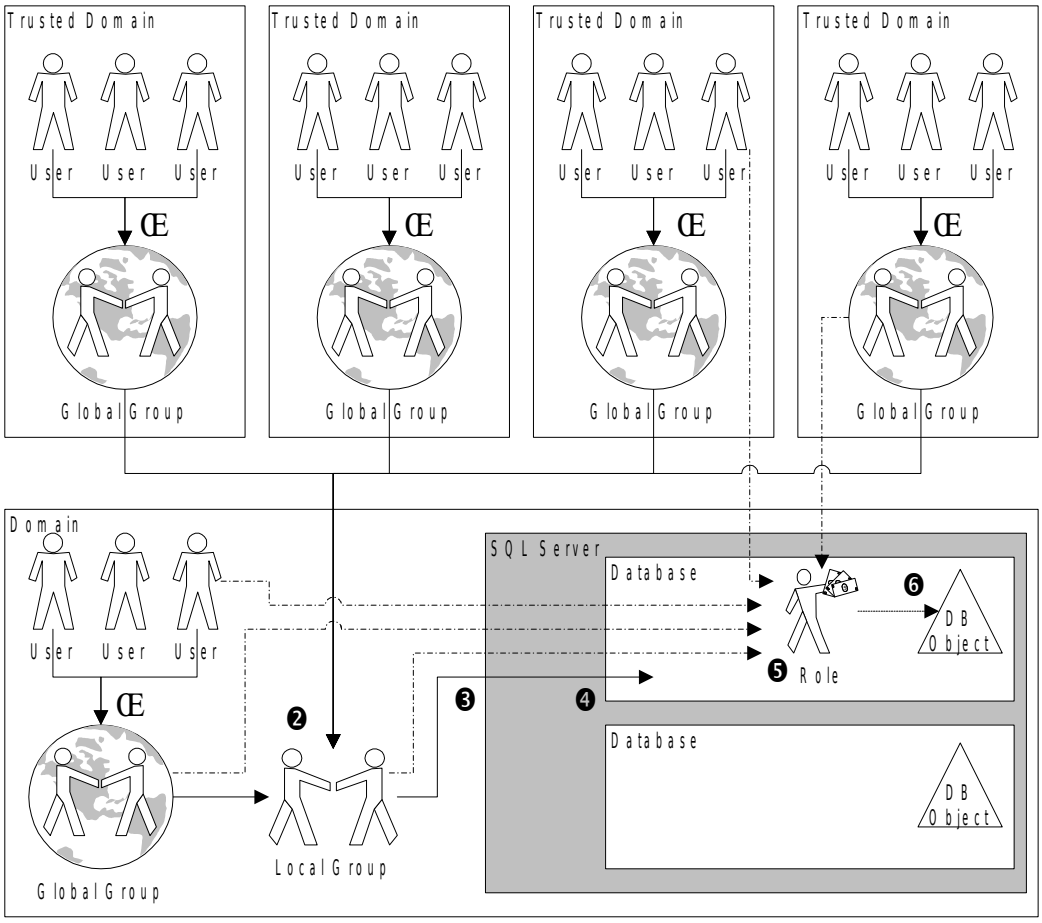
2. The Windows NT global groups from the various domains are placed into a Windows NT local group.

3. Windows NT local group is granted rights to log in to SQL Server 7.0.

4. The Windows NT local group is granted access rights to the appropriate databases. This Windows NT local group may not be the same one as was used for granting login rights in Step 3. Therefore, Steps 1 and 2 are often repeated to group the users by access permissions required.

5. The Windows NT local group is assigned permissions on the specific database objects.

The other approach to security is based on the use of roles, and is usually implemented in the following way.



When using roles to assign object permissions, individuals must still be granted permissions on the server and the database using the recommended approach.

Steps 1 through 4 are the same for both diagrams, except that multiple Windows NT global and local groups would probably not be created.

5. Individual Windows NT accounts and Windows NT groups are assigned to a role.

6. Object permissions are then assigned to the roles.

Roles reduce the requirements of grouping users within Windows NT, by grouping the users within SQL Server 7.0.

Securing Access to the Server

Access to the server is controlled differently by the two SQL Server 7.0 security modes. However, after a user gains access to the server, the security mechanisms are identical.

SQL Server 7.0 security defaults to Mixed Mode when it is installed. However, when SQL Server 7.0 is installed on the Windows NT operating system, Windows NT Authentication Mode may be selected, and is strongly recommended.

Windows NT Authentication Mode

In this mode, administrators create users and groups in the Windows NT User Directory, and then grant those users and groups permissions to log in to SQL Server 7.0.

Windows NT Level

At the Windows NT level, administrators should create a login account for each user who will be accessing SQL Server (if the user does not already have an account).

In each user accounts domain, global groups should be created to group users by job requirements. The users should then be placed into the appropriate global groups in their domain.

On the computer running SQL Server 7.0, local groups should be created according to the various job requirements for which access to SQL Server needs to be granted. The appropriate global groups from the various trusted domains should then be placed into the respective local groups on the computer running SQL Server.

The global group and local group requirements outlined earlier may seem like a lot for small, single domain networks; however, experience has shown that there is usually great value in doing this.

The base requirement here is to get all the users with the same security requirements grouped into one unit, which can then be used by the database administrator to grant access to SQL Server 7.0. Granting access to SQL Server by group does not eliminate the ability to identify the individual user from within a database.[7]

---

[7] This is largely the same action as securing a file on the NTFS file system by giving access to members of the **SALES** group only. If Bob, a member of the **SALES** group, tries to access the file, the audit log will contain an entry for Bob, not **SALES**.

Although the recommendations are strong, the database administrator is able to assign permissions to objects for Windows NT global groups, local groups, and individual user accounts.

Programmatically creating user accounts and groups in the Windows NT environment is beyond the scope of this document. This can be achieved by using the ADSI object model from Microsoft Visual Basic, or by interfacing directly to the Win32 API from C++.

Server Level

At the SQL Server 7.0 level, permissions must be granted for the created Windows NT local groups to log in to SQL Server. Permission to log in to SQL Server can also be granted to users directly, but is not as practical to administer except for the smallest of environments.

Permissions to log on to the server can be granted through the user interface or implemented programmatically using Microsoft Visual Basic or Transact-SQL.[8]

Note: This document is not intended to be a walk-through of the user interface; hence, the implementation of security from a user interface perspective will not be outlined.

New stored procedures have been written to allow the granting of access for Windows NT users and groups. The following Transact-SQL statement

```
/* Grant Login */
sp_grantlogin 'REDMOND\SALESLG'
```

grants login rights to the **SALESLG** local group:

Alternatively, login rights can be granted with the following Visual Basic code:

[8] the following definitions within the context of this document. Visual Basic refers to writing an application using the Visual Basic environment with the SQL-DMO library or using any Visual Basic for Applications environment with the SQL-DMO library. Transact-SQL (Transact Structured Query Language) refers to the type of SQL implemented in SQL Server 7.0

```
' Declare variables
Dim oServer As SQLDMO.SQLServer
Dim oLogin As SQLDMO.Login

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Create the Login object
Set oLogin = CreateObject("SQLDMO.Login")

' Set the appropriate properties
oLogin.Name = "REDMOND\SALESLG"
oLogin.Type = SQLDMOLogin_NTGroup

' Add the Login object to the server's Logins collection
oServer.Logins.Add oLogin
```

Mixed Mode

In the same way as Windows NT Authentication Mode used the Windows NT User Directory, Mixed Mode is also able to use the Windows NT User Directory. However, in Mixed Mode, users may also supply a username and password pair, which SQL Server 7.0 uses to authenticate the user and grant access. This mode is recommended only when SQL Server 7.0 is installed on Windows 95 or Windows 98 operating systems, or for backward compatibility with existing applications.

Server Level

In order to allow a user access to SQL Server 7.0 using non-trusted connections, user accounts must be created on the SQL server.

The following Transact-SQL script creates a login for a non-trusted

```
/* Add a Login */
sp_addlogin
'Bob',
'password',
'pubs'
```

connection:

This statement adds a user called Bob and sets the password to **password**. The default database becomes **pubs**. The default database is the database to which the user is switched when she or he attempts to log in. A user must still create a user account in the default database for this to work; **sp_addlogin** does not add a user account in the referenced database.

Alternatively, the above can also be achieved using Visual Basic as follows:

```
' Declare variables
Dim oServer As SQLDMO.SQLServer
Dim oLogin As SQLDMO.Login

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Create the Login object
Set oLogin = CreateObject("SQLDMO.Login")

' Set the appropriate properties
oLogin.Name = "Bob"
oLogin.Type = SQLDMOLogin_Standard
oLogin.SetPassword “”,“password”

' Add the Login object to the server’s Logins collection
oServer.Logins.Add oLogin
```

Note: When SQL Server 7.0 is installed on Windows NT and configured to use Mixed Mode, trusted connections can still be made by capable clients.

Fixed Server Roles

To add users to the fixed server roles, use the following Transact-SQL

```
/* Add Bob to the sysadmin server role */
sp_addsrvrolemember "REDMOND\Bob", "sysadmin"
```

statement:

Windows NT users and groups can be added to server roles.

The following code shows how a user is added to a server role using SQL-

```
' Declare variables
Dim oServer As SQLDMO.SQLServer

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Add Bob to the sysadmin server role
oServer.ServerRoles("sysadmin").AddMember ("REDMOND\Bob")
```

DMO:

Securing Access to the Database

Successful login does not automatically allow a user to access all databases on SQL Server 7.0. Permissions must be granted to allow users to access a database.

In the context of this section, we do not differentiate between non-trusted users, Windows NT users, and Windows NT groups.

Note: When reference is made to Windows NT user or groups, they can also be users or global groups in trusted domains.

Within each database, a user is created and is linked to a SQL Server login, a Windows NT user, or a Windows NT group.

Note: SQL Server Enterprise Manager (which is a Microsoft Management Console (MMC) snap-in for administering SQL Server 7.0) does not allow the creation of users who do not have specific logon permissions. The MMC creates a list of all accounts that have been granted the permission to log on to the server, and a selection needs to be made from this list. The same applies to the SQL-DMO object model.

With Transact-SQL, any valid SQL Server logon, Windows NT user, or Windows NT group can be granted the rights to access the database, whether a specific logon exists in the **sysxlogins** table in the **master** database.

Although not a technical requirement, if you are using trusted connections it is strongly recommended that you create users with the same username in each database as the logon name.

Some examples for the Transact-SQL statements required to grant

```
/* Grant access to Bob */
sp_grantdbaccess
'REDMOND\Bob'

/* Grant access to
Wendy, referring to her
by first name within this
database */
sp_grantdbaccess
'REDMOND\WendyH',
'Wendy'
```

permission to use a database are:

Note: Only one modification would be required to make this example work with non-trusted clients. Instead of the domain username, use the username that SQL Server 7.0 uses to authenticate the user.

Using SQL-DMO, the equivalent functionality would be achieved by the

```
' Declare variables
Dim oServer As SQLDMO.SQLServer
Dim oUser As SQLDMO.User

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Create the User object
Set oUser = CreateObject("SQLDMO.User")

' Set the appropriate properties
oUser.Name = "Bob"
oUser.Login = "REDMOND\Bob"

' Add the User object to the servers Users collection
oServer.Databases("pubs").Users.Add oUser
```

following code:

Securing Access to the Database Objects

In SQL Server 7.0, permissions can be assigned to allow user to execute certain statements, and to access certain database objects. Statement permissions restrict who can execute statements such as CREATE DATABASE, CREATE TABLE, or CREATE VIEW. Object permissions restrict access to objects such as tables, views, or stored procedures.

Object permissions are dependent on the object being referenced. For example, object permission for tables include the SELECT, INSERT, UPDATE, DELETE, and REFERENCES permissions, while the object permissions on a stored procedure include EXECUTE permissions.

Permissions can be granted to roles and users.

User-Defined Database Roles

In an ideal environment, roles would not be necessary. In such an environment, all users would be running SQL Server 7.0 on Windows NT in Windows NT Authentication Mode. The database administrator could ask the Windows NT administrator to place all the users with a specific data access requirement (or role) into one Windows NT group, and the database administrator would then grant permissions to that Windows NT group as required.

However, as this is not the case in most environments, creating Windows NT groups is not always possible. For example, when installing SQL Server 7.0 on the Windows 95 or Windows 98 operating systems, Windows NT groups are not technically possible. In this case, roles can be used to group users by their permission requirements.

Any Windows NT user or group can be assigned to a role, which can then be assigned permissions to database objects the same way as database users are assigned permissions.

Note: User-defined roles can be created only in a database. Fixed server roles and fixed database roles are predefined and cannot be modified.

```
/* Add role for Telephone Operators */
sp_addrole "TelephoneOperators"
```

Roles can be created with the following Transact-SQL code:

Alternatively, roles can be created with the following Visual Basic code:

```
' Declare variables
Dim oServer As SQLDMO.SQLServer
Dim oDbRole As SQLDMO.DatabaseRole

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Create the Database Role object
Set oDbRole = CreateObject("SQLDMO.DatabaseRole")

' Set the appropriate properties
oDbRole.Name = "TelephoneOperators"

' Add the Role object to the servers Role collection
oServer.Databases("pubs").DatabaseRoles.Add oDbRole
```

After a user-defined database role is created, it makes sense to add users, groups or other roles to it. Roles can be nested, although not in a circular manner, as this would not be productive.

This example Transact-SQL code adds a Windows NT user, a Windows NT group, and a database role to the newly created role:

```
/* Add a Windows NT user to the TelephoneOperator's role */
sp_addrolemember "TelephoneOperators", "REDMOND\Bob"

/* Add a Windows NT group to the TelephoneOperator's role */
sp_addrolemember "TelephoneOperators", "REDMOND\Sales"

/* Add HelpDeskOperator's role to TelephoneOperator's role
*/
sp_addrolemember "TelephoneOperators",
"HelpDeskOperators"
```

```
' Declare variables
Dim oServer As SQLDMO.SQLServer

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("MSNZBENTHOM")

' Use with statement for code legibility
With oServer.Databases("pubs").DatabaseRoles("TelephoneOperators")

    ' Add the Windows NT user to the TelehoneOperator's role collection
    .AddMember ("REDMOND\Bob")

    ' Add the Windows NT group to the TelehoneOperator's role collection
    .AddMember ("REDMOND \Sales")

    ' Add the HelpDeskOperators role to TelehoneOperator's role collection
    .AddMember ("HelpDeskOperators")

End With
```

And again with SQL-DMO:

Granting and Denying Permissions to Users and Roles

Permissions within a database are always granted to database users, roles, and Windows NT users or groups, but never to SQL Server 7.0 logons.

There are three methods used to set the appropriate permissions for users or roles within a database: granting permissions, denying permissions, and revoking permissions.

In this section, each of these methods will be applied in a Visual Basic example and in a Transact-SQL example. The effective user permissions that are the result of implementing these statements are outlined in "Permissions", and are explained further in "How Permissions Are Checked." The following Transact-SQL code grants Bob and Jane permissions to SELECT from the **authors** table, and grants Jane permissions to INSERT into the **titles** table:

```
/* Grant permissions to SELECT */
GRANT SELECT
ON authors
TO Bob, [REDMOND\Jane]
GO

/* Grant permissions to INSERT */
GRANT INSERT
ON titles
TO [REDMOND\Jane]
GO
```

The previous example shows how the GRANT statement works when permissions are granted to explicit users of the database (Bob) and when permissions are granted to a Windows NT user (Jane).

Note the syntax used when referring to a Windows NT account (user or group), as opposed to a SQL Server 7.0 login. The square brackets are used as delimiters throughout SQL Server 7.0 in much the same way as double quotation marks are used. In this case, the square brackets (or quotation marks) were necessary because the backslash (\) character is not a legal character in SQL Server names. The square brackets would also be necessary if the user had a space, or any other illegal character, in

```
' Declare variables
Dim oServer As SQLDMO.SQLServer

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Grant Jane and Bob the right to select from authors table
oServer.Databases("pubs").Tables("authors").Grant SQLDMOPriv_Select, "Bob"
oServer.Databases("pubs").Tables("authors").Grant SQLDMOPriv_Select, _
      "[REDMOND\Jane]"

' Grant Jane the right to select from authors table
oServer.Databases("pubs").Tables("authors").Grant SQLDMOPriv_Select, _
      "[REDMOND\Jane]"
```

his/her name. The same example would be coded like this in Visual Basic:

In the previous examples, there is little difference between granting access to a user by fully qualifying their domain name, and granting access to a user who already has permissions to directly access the database. Due to these similarities, the following examples will show the code only for existing database users.

The following Transact-SQL statement shows how a user can be denied SELECT permissions:

```
/* Deny permissions to SELECT */
DENY SELECT
ON authors
TO Bob
GO
```

And again using Visual Basic:

```
' Declare variables
Dim oServer As SQLDMO.SQLServer

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Deny Bob the right to select from authors table
oServer.Databases("pubs").Tables("authors").Deny SQLDMOPriv_Select, "Bob"
```

Here is a Transact-SQL example that shows how to revoke permissions

```
/* Revoke permissions to SELECT */
REVOKE SELECT
ON authors
FROM Bob
GO
```

from a user:

```
' Declare variables
Dim oServer As SQLDMO.SQLServer

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Revoke Bob the right to select from authors table
oServer.Databases("pubs").Tables("authors").Revoke SQLDMOPriv_Select, "Bob"
```

The Visual Basic version of this code would be as follows:

It is important to understand that revoking permissions is not the same as denying permissions. Revoking permissions deletes a previous grant or deny, while denying permissions prohibits access even when access permissions have been granted. For more information, see “Deny.”

Ownership Chains

Although ownership chains are not new to SQL Server 7.0 security, they are a concept that very few administrators really understand. A thorough understanding of ownership chains is crucial for the development of a secure SQL Server 7.0 environment.

The concept of ownership chains is established when permissions on an object are checked. For example, when a user accesses a view, the permissions on the view should be checked. But what about the permissions on the underlying table?

SQL Server 7.0 always checks the permissions on objects when there is a broken ownership chain. A broken ownership chain is when an object does not have the same owner as its underlying objects. For example, if Bob creates a table, and Mary creates a view based on that table, there is a broken ownership chain.

In relation to security, broken ownership chains specify where permissions should be checked over and above the original object accessed. This makes for a very practical model.

The concepts of ownership chains are best explained with a detailed example. Assume that Bob owns a table. He secures access to it by granting SELECT permissions on the table to only Mary. Mary creates a view of Bob's table that suits her needs. One day Sue sees Mary using this view, and exclaims how brilliant it is. Mary agrees to give Sue access to the view. It was not Bob's original intention that Sue should see the data in his table. Fortunately, there is a broken ownership chain, as Bob owns the table and Mary owns the view. The owner of the view does not own the underlying objects. In this case, when Sue tries to use the view, SQL Server checks the permissions on the view, to ensure that Sue has been granted access. Following this, the permissions on Bob's table are also checked. If Sue has not been granted access to the table, she cannot use the view. This is because of the broken ownership chain. Effectively, a broken ownership chain guards against a user hijacking the access permissions.

Conversely, if Bob decided to create the view and to deny Sue access to his table, but grant her access to the view, Sue would be able to access the view. This is because the permissions are checked only when Sue accesses the view. There is no broken ownership chain, so permissions for the underlying table are not checked. Because Bob has created both objects, he should understand that giving access to the view requires implicit access to the underlying objects.

An example where Server 7.0 implements the power of ownership chains is in the implementation of passwords. Users are not allowed to update the system tables directly, especially not those found in the **master** database. When using SQL Server 7.0 in Mixed Mode, username and password combinations are stored in the **sysxlogins** system table. Users should be given the opportunity to change their passwords on a regular basis. SQL Server 7.0 achieves this by implementing a stored procedure to change the password, which any user can execute. Access to the **sysxlogins** table is denied, but permission to execute the **sp_password** stored procedure is granted to all users. As the **sp_password** stored procedure and **sysxlogins** system table have the same owner, there is no broken ownership chain, and permissions are checked only on the stored procedure.

Ownership chains allow SQL Server 7.0 to implement a security system that allows the owner of the original data to maintain control over who is authorized to access it. At the same time, performance is increased because permissions do not require checking if the ownership chain is not broken.

Securing Access Through Application Roles

This section outlines how an application can make use of application roles. To use application roles, perform the following steps:

1. Application role must be created.
2. Application role must be assigned permissions.
3. Client application connects to SQL Server 7.0.

4. Client application activates the application role.

The first two steps of this process are usually separated from the last two. Therefore, two code fragments will follow for Transact-SQL and Visual Basic respectively.

```
/* Create the app role */
EXEC sp_addapprole
"AccAppRole", "ABC"

/* Grant permissions to SELECT */
GRANT SELECT
ON authors
TO AccAppRole
GO
```

The Transact-SQL script is as follows:

```
/* Activate the role */
EXEC sp_setapprole
“AccAppRole”,
{ENCRYPT N “ABC”}
```

And the code to activate the role:

The encryption of the password is optional, but ensures greater security when the password has to pass through a wide area network (WAN).

```
' Declare variables
Dim oServer As SQLDMO.SQLServer
Dim oDbRole As SQLDMO.DatabaseRole

' Create a server object and connect
Set oServer = CreateObject("SQLDMO.SQLServer")
oServer.Connect ("SERVERNAME")

' Create the Role object
Set oDbRole = CreateObject("SQLDMO.DatabaseRole")

' Set the appropriate properties
oDbRole.Name = "AccAppRole"
oDbRole.AppRole = True
oDbRole.Password = "ABC"

' Add the Role object to the servers Role collection
oServer.Databases("pubs").DatabaseRoles.Add oDbRole
```

And the same again in Visual Basic:

```
' Declare variables
Dim oConnection As ADODB.Connection

' Create the connection object and connect
Set oConnection = CreateObject("ADODB.Connection")
oConnection.Provider = "sqloledb"
oConnection.Open "Server=SERVERNAME;Database=pubs;Trusted_Connection=yes"

' Activate the application role - No error handling for this demo
oConnection.Execute "EXEC sp_setapprole 'AccAppRole', {ENCRYPT N 'ABC'}, 'ODBC'"
```

And to use the role:

The encryption style (last parameter) must be set for OLEDB and ODBC data sources. All other data sources cannot explicitly encrypt the

password. In these cases, it is imperative to use an encrypted communications protocol with the server.

A point often overlooked is that application roles are implemented per session. If your application opens multiple sessions and all sessions are required to use the same role, each session must first activate the role.

This can be used to provide much more granular security than ever before. For example, a client application could use the user's security context on some connections, while using an application role on another.

Note: When using application roles, executing SELECT USER returns the name of the application role currently being used. If the identity of the logged-on user is required, use the following SQL statement: SELECT

SYSTEM_USER.

Implementation of Server Level Security

Use of SIDs

Logon security over trusted connections was handled in an indirect way in SQL Server 6.5 and earlier. Users and administrators were granted access to particular registry keys[9] on the computer running SQL Server, and if they had permissions to access the key, they were allowed into SQL Server as a user or **sa** (Full Control = **sa**, Read-Only = ordinary user). SQL Security Manager was the tool that configured the security on the registry key, and placed the fully qualified usernames into the **syslogins** system table.

When a Windows NT user or group is either granted or denied access to SQL Server 7.0, this information is stored in the **sysxlogins** system table. Permissions on a registry key no longer control access to the server. SQL Server 7.0 identifies users connecting through a trusted connection by their SID and group membership SIDs.

SQL Server 7.0 checks whether the user's SID or group membership SIDs have been specifically denied access to the server. If so, the user will not be granted access to the server. If the user is not specifically denied access, the server checks whether the user has been granted access directly or by virtue of a group membership. If access has been granted, the connection to SQL Server 7.0 is maintained. Then, the user proceeds to the appropriate default database (where the user must also have been granted access). The user's access rights are then checked for any objects that are attempted to be accessed. If access has not been granted for a particular set of logon credentials, the connection to the server is terminated.

[9] The key is: HKEY_LOCAL_MACHINE\Software\Microsoft\MSSQLServer\MSSQLServer

Generation of GUIDs for Non-Trusted Users

For non-trusted connections, such as when SQL Server 7.0 is installed on a Windows 95 or Windows 98 operating system, Windows NT SIDs are not available. In this case, SQL Server 7.0 generates a 16-byte globally unique identifier (GUID). The generated GUID is then used internally in the same way as Windows NT SIDs are used for Windows NT users and groups. In this way, security can function identically in a trusted and non-trusted environment.

Renaming Windows NT User or Group Accounts

When a Windows NT user or group is renamed using the User Manager for Domains tool, SQL Server 7.0 is unaware of that change. SQL Server 7.0 maintains the user or group's fully qualified name in the **sysxlogins** table for performance reasons, as it can be very slow to query the domain controller for this information. This is particularly true when a lot of name lookups are done or the domain controller is connected over a slow WAN link.

The fact that the names of users and groups may be different from SQL Server 7.0 to Windows NT does not cause any security problems. The permissions set for the user or group continue to function correctly, as SQL Server relies only on the SIDs internally.

When the SUSER_SNAME and SUSER_SID functions are used to return the username and user's SID respectively, they attempt to resolve it by first querying the **sysxlogins** table. The Windows NT Local Security Authority (LSA)[10] is queried only if the **sysxlogins** table does not contain the username or SID.

Another effect of this is that the usernames in system messages may not report an up-to-date name.

sysxlogins System Table

The **sysxlogins** system table contains the login permission (or lack of) for users.[11]. This system table exists only in the **master** database.

SQL Server 7.0 ships with three views that depend on the **sysxlogins** table. The **syslogins** view provides backward compatibility, while at the same time interprets the **status** column so that it can be understood more readily. The **sysremotelogins** view provides backward compatibility, and allows information regarding remote logins to be accessed more readily. Finally, the **sysoledbusers** view provides information regarding remote logins. For more information, see "sysoledbuser View."

---

[10] For additional information, see the *Windows NT Resource Kit*.

[11] In SQL Server version 6.5 and earlier, this information was stored in the **syslogins** system table. **syslogins** can still be queried, as it is a view over the **sysxlogins** table for backward compatibility. **Note:** This view should not really be necessary, as system tables should not be accessed directly. System tables may change at any time.

xstatus Column

The **xstatus** column provides a number of status settings, including the server role memberships. The various status values are listed in the following table. For more information, see “Predefined Roles,” “hasaccess and “denylogin

Purpose	bit[12]	Notes
Denylogin	1	
Hasaccess	2	
Isntname	3	Not “ISN’T”, but “IS WINDOWS NT”
Isntgroup	3	Only if status bit 4 is not set
Isntuser	4	Must also have status bit 3 set
Sysadmin	5	Server role
Securityadmin	6	Server role
Serveradmin	7	Server role
Setupadmin	8	Server role
Processadmin	9	Server role
Diskadmin	10	Server role
Dbcreator	11	Server role

dbid and language Columns

An area that is often misunderstood is how a user gets the default database and default language settings. When a user connects to SQL Server 7.0, the server looks for a row containing the user’s specific SID (or GUID in the case of non-trusted connections) in the **sysxlogins** table. If found, the default database and default language settings are taken from the found row. If one was not found, the server looks for SIDs that belong to the groups of which this user is a member. The default database and default language settings from the first group that is found are used. This implementation is specific to each default.

If the first found group (of which the user is a member) contains a default language, but the default database is NULL, SQL Server continues to the next group (of which this user is a member), and tries to ascertain the default database from there.

For example, if Bob is a member of the **SALES** and **MARKETING** groups, and he does not have a default database and default language configured specifically for his account, the system looks for the default settings applied to the **SALES** and **MARKETING** groups. The first settings returned are used.

Thus, if a user is a member of more than one group, and does not have default database and default language settings assigned, the defaults chosen are not guaranteed.

---

[12] Bit: counting status bits from right to left.

It is possible to assign default database and language settings specifically to a user without granting specific login rights to that user. The user would be permitted access to SQL Server based on group memberships, but would receive the default settings based on the defaults recorded in the **sysxlogins** system table specifically for that user. In this case, the **hasaccess** flag of the **sysxlogins** table would be set to zero for this user's specific entry in the **sysxlogins** table.

hasaccess Status

The **hasaccess** status in the **sysxlogins** system table is used to allow default settings to be configured for a specific user without implicitly granting that user access. Remember that the **sysxlogins** table is usually used to grant login rights to particular users or groups. If the **hasaccess** status is zero, the user is not granted login access explicitly. However, when the user logs on (through a group membership), the defaults are established.

The **hasaccess** status is also crucial for another reason, which is best explained by virtue of an example. Bob is a member of the **REDMOND\SALES** group, and has not been specifically granted permission to log in to SQL Server 7.0. There are no entries for Bob in the **sysxlogins** table. However, the **REDMOND\SALES** group has been granted login permissions, so Bob is able to log in. When Bob becomes a member of a fixed server role, he should not automatically gain permission to access the SQL server directly; his access should still be through the **SALES** group. In this case, a new row is added to **sysxlogins** for Bob, but the **hasaccess** flag is set to zero so that the relevant server role memberships can be granted, without implicitly granting access to the server.

Another situation where the **sysxlogins** table contains entries that are not specifically granting access to a user or group is when the **denylogin** flag is set.

denylogin Status

The **denylogin** status is used to mark a user or group as being explicitly denied access to SQL Server 7.0. For example, if a specific user (or group) should have their access to SQL Server prevented, the following Transact-

```
sp_denylogin 'REDMOND\Bob'
```

SQL statement can be executed:

```
sp_revokelogin 'REDMOND\Bob'
```

This is not the same as:

The difference between the two statements is that the first statement denies access to SQL Server, the second revokes access using that particular account. If the user is a member of the **MARKETING** group, which does have access, the second statement would still allow Bob to continue accessing the server by virtue of membership in the **MARKETING** group. The first statement would deny access irrespective of any group memberships that may grant access.

Note: On Windows NT, one DENY is all that is required to lock a user out of a resource.

sysremotelogins View

The **sysremotelogins** view is provided for backward compatibility. In SQL Server 6.5 and earlier, a table with the name **sysremotelogins** provided the mapping used for remote logins.

sysoledbuser View

When a user wants to run a query on a remote server, the local server must log on to the linked server on behalf of the user. The **sysoledbuser** view is new to SQL Server 7.0 and provides this information.

The **sp_addlinkedsrvlogin** stored procedure is used to add new linked accounts to remote servers. This information is stored in the **sysxlogins** table. The stored procedure requires the remote server name, the local username, the remote username, and the remote password as parameters. An extension has been developed for Windows 2000 Professional, which will provide security account delegation functionality.

Implementation of Object-Level Security

How Permissions Are Checked

SQL Server 7.0 uses SIDs to identify Windows NT users and groups. However, due to the length of SIDs (up to 85 bytes), SQL Server 7.0 maps the SIDs to user IDs inside each database. The SIDs are mapped to user IDs in the **sysusers** table. The user ID is then used in the **sysobjects** table to denote the owner of a table. It is also used in the **sysprotects** table to set permissions on objects, and in the **systypes** table to denote the owner of a user-defined type.

When a user connects to SQL Server 7.0, the server creates a Process Status Structure (PSS)[13] structure in memory, which is comprised of the user's SID, group SIDs, and other security and state information. This structure is a snapshot taken when the user connects, and the snapshot is not refreshed. This structure exists per session connecting to the server; a single user establishing multiple sessions with SQL Server 7.0 will have multiple PSS structures.

When the user proceeds to access a database, SQL Server checks the **sysusers** table to ascertain whether the user has been denied access directly or by virtue of being a member of a group that has been denied access. If the user is denied access, this is enforced, otherwise the **sysusers** table is checked again, but this time all user IDs are collected for which the user qualifies. After it is established that the user has been granted access to the database, the **sysmembers** table is scanned, so that all the role memberships of the user can be established. For example, the user may be a member of a role, a member of a Windows NT group or aliased to another user. The user IDs of all the applicable memberships

[13] This structure is referred to in several error messages produced by SQL Server 7.0.

are established so that the appropriate permissions for this user can be applied. Unlike the PSS structures, this information is not persisted.

When the user starts accessing objects in the database, the applicable permissions are determined by checking the **syspermissions** table for entries with matching user IDs (as identified earlier). The system checks DENY permissions first, and if found, the user will not get the requested access to the object. However, if no DENY permissions are found, and entries that give the user the required access exist, access is granted. The effective access permissions are then cached so that repeated access to the same objects by the same user do not continue to incur the cost of checking the access permissions.

Cost of Changing Permissions

As stated earlier, SQL Server 7.0 caches the object permissions on a per-session basis in order to avoid incurring the cost of checking permissions for repeated access of the same objects. Unlike the PSS, which does not change the security information after it is created, the permissions cache is always up-to-date. This implemented by the versioning method.

When the initial checking of permissions takes place, a version number is established. When the permissions on an object are changed, SQL Server 7.0 increases the version counter. Whenever an object is accessed, the version of the permission counter is checked, and if it differs from the cached counter, the content of the cache is discarded, and the effective permissions are reestablished.

The cached security is used whenever an object is accessed, providing that the version counter has not changed. If the counter has changed, a small overhead is incurred for that operation.

Renaming Windows NT User or Group Accounts

With SQL Server 7.0 it is possible to grant Windows NT users and groups permissions to access objects in the database directly. In that case, the SID and Windows NT user or group names are stored in the **sysusers** table.[14]

When the Windows NT administrator renames the Windows NT group or user, the name change is not propagated to SQL Server 7.0.

Despite the fact that this seems to be a significant problem, it is actually more of a solution to what would otherwise become chaos.

With SQL Server 7.0, as with previous versions, administrators and developers are writing numerous stored procedures, Transact-SQL scripts, triggers, and so on. Assume that Susie Jones is a user who creates a table in the database. Susie's login name is SUSIEJ, and her table is named **SUSIEJ.SALESDEMO**. Susie grants permissions for others to access her table, and several of her colleagues create views and stored procedures based on her table. When Susie gets married to Bob Taylor, her username is renamed to SUSIET. If SQL Server 7.0 were to pick up the change, Susie's table would suddenly be **SUSIET.SALESDEMO**, which is a completely different object. The views, stored procedures, and any code that was written to access this table would break.

In the interest of stability, SQL Server 7.0 does not automatically rename user accounts when the real account in the Windows NT User Directory is renamed.

---

[14] Refer to "Renaming Windows NT User or Group Accounts" for a discussion of the same problem with login names.

sysprocedures System Table Removed

In SQL Server 6.5 and earlier, when a stored procedure was created, the query text was stored in the **syscomments** system table, and a normalized query tree was saved in the **sysprocedures** system table. The normalization process parsed the SQL statements into more efficient formats and resolved all referenced objects into their internal representations. When the procedure was subsequently executed, the tree was retrieved from **sysprocedures** table and used as the basis for an optimized execution plan, which was then stored in the procedure cache.[15]

At first it may appear that there is no connection between the process outlined earlier and security. However, the relevance of this to security is based on the fact that some software developers[16], in an attempt to protect their source code, deleted the original SQL text from **syscomments**. In most cases, the original SQL text really was not used again until the server was upgraded with a later version of SQL Server, or when a service pack was applied. Microsoft provided a better mechanism for hiding the original SQL text from anyone who should not have access with the introduction of the WITH ENCRYPTION option in SQL Server version 6.0. This option encrypted the original SQL text upon the creation of the stored procedure.

With SQL Server 7.0, any administrator who deletes the appropriate entries from the **syscomments** table will find that the stored procedure no longer executes. This is because the **sysprocedures** table has been removed from SQL Server 7.0, which now obtains the SQL text directly from the **syscomments** table prior to execution.

With Grant Option

The WITH GRANT OPTION is optional syntax that can be used with the GRANT statement. Although this existed in SQL Server 6.5, it is included here because few people actually understand it.

This option applies only to object permissions, and provides the recipient of the GRANT statement with the ability to pass on that permission.

For example, if Bob granted Jane SELECT permissions and used the WITH GRANT OPTION, Jane would be able to grant SELECT permissions to some of her friends.

When Bob revokes SELECT permissions from Jane, he can use the CASCADE option to revoke the SELECT permissions from those friends to whom Jane had granted SELECT permissions.

---

[15] In fact, this approach was also used for views, defaults, rules, triggers, check constraints and default constraint

[16] This is particularly true prior to SQL Server 6.0.

sysusers System Table

In some ways, the **sysusers** table is to the database what the **sysxlogins** table is to the server. The **sysusers** table exists in each database, and contains information about who is granted or denied access to the database.

hasdbaccess column

The **hasdbaccess** column is used in a similar way to the **hasaccess** column in the **sysxlogins** table. In this case, however, entries with this flag set to zero are created when a user: has not been granted rights to access the database explicitly but creates objects; is explicitly granted permissions; or is added explicitly to a role. Objects created by a user are always owned by the user, and are not owned by the group through which the user was granted access to the database.[17] An exception is when a user, who is a member of a role or Windows NT group, explicitly qualifies the role or group as the object's owner when the object is created.[18] In this situation, an entry for the user must exist in the **sysusers** table so that the object can have the appropriate owner. The entry is created automatically, but the user does not get explicit access to the database automatically because the **hasaccess** flag is set to zero.

Roles, which are also listed in the **sysusers** table, have the **hasdbaccess** column set to zero.

sysmembers System Table

The **sysmembers** system table is one of the smaller tables. It contains two columns, and is used to record the membership of users in database roles. It contains one row for each member of a database role.

SQL Server 7.0 improves the performance related to roles by placing a user's first membership of a role into the **gid** column of the **sysusers** table. Thus, when SQL Server 7.0 tries to identify all the roles to which a member belongs, it does not have to query the **sysmembers** table if the **gid** column of the **sysusers** table contains zero. If the entry in that column is not zero, the entry specifies one of the roles and the **sysmembers** table must be queried for a complete list of all the roles to which the user belongs.

syspermissions System Table

The **syspermissions** system table, which exists in every database, is new to SQL Server 7.0. In previous versions, the **sysprotects** system table was used to maintain all permissions. Now the **syspermissions** table is used

---

[17] This makes sense, as a user may be a member of more than one group. Which group should become the owner of the object?

[18] The following syntax is legal:

```
CREATE TABLE [builtin\administrators].test_table(acolumn
VARCHAR(2))
```

to track permissions that have been granted or denied to users. For more information about backward compatibility with the **sysprotects** table, see "**sysprotects** System Table."

The **syspermissions** system table consists of very few columns. The **id** column references the object ID for which the permissions are being granted or denied. For statement permissions, this column is set to zero.

The next two columns, **grantee** and **grantor**, are self-explanatory. The value used here is the ID of the role, Windows NT user, or Windows NT group as it is found in the **sysusers** table.

The **actadd** column refers the positive permission (or permissions granted) on all columns (in the case of a table) of the object, while the **actmod** column refers to the negative (or permissions denied) permissions on all columns (in the case of a table) of the object.

The remaining columns are used only when column-level permissions are implemented. The **seladd** column is for SELECT granted permissions, and is a bitmap of the columns that have been granted this permission. As column IDs are never reused, the bitmap approach works very well. The **selmod** column is for SELECT permissions denied.

The next two columns are implemented in the same way as the previous two, except that they apply to UPDATE permissions.

The last two columns refer to the REFERENCES permissions, and are implemented in the same way as the previous four columns.

sysprotects System Table

The implementation of the **sysprotects** system table has changed. In SQL Server 7.0 and earlier, the **sysprotects** table stored the object permissions. In SQL Server 7.0, this information is now stored in the **syspermissions** table.

In most cases, where the implementation of a system table changed, Microsoft provided a view that provided backward compatibility. One example of this is the implementation of the **master..sysxlogins** table; the **syslogins** view is now provided for backward compatibility.

In the case of the **sysprotects** system table, however, the change in the underlying system table was much more major, and a view was not able to provide backward compatibility efficiently. For this reason, Microsoft opted not to implement a **sysprotects** view, but to create a special table named **sysprotects**, which appears as a normal table to the system, but is really created dynamically when required.

The **sysprotects** table is therefore somewhat like a view, as it does not actually have any persisted data pages. However, because the view-like behavior is implemented in the database engine, the **sysprotects** object appears as a table in the **sysobjects** table.

Named Pipes and Multiprotocol Permissions

When discussing the internal security of SQL Server 7.0, it is important to point out a key concept that is often overlooked. This is not new for SQL Server 7.0, but is mentioned here for completeness of understanding.

The Named Pipes Net-Library is an inter process communications (IPC) mechanism, which is implemented over the IPC$ share on Windows NT. Thus, when a client connects to SQL Server using the Named Pipes Net-Library, the connection is made to the IPC$ share, at which point authentication takes place. This was mentioned in passing during the discussion of the new security features. For more information, see “Networking Protocols.” After Windows NT has authenticated the client (in the same way as it would for access to any other resource), the Named Pipes session is established over the IPC$ share. This takes place before any attempt is made to pass the connection to SQL Server.

It is important that all users who will be connecting to SQL Server 7.0 using the Named Pipes Net-Library have a Windows NT account and have Windows NT permissions to access the IPC$ share. If you do not want this authentication to take place, switch to another network library such as TCP/IP Sockets or Multiprotocol, as these connections are not validated against the Windows NT IPC$ share.

When using the Multiprotocol Net-Library, Windows NT authentication also takes place before SQL Server 7.0 passes the connection. This is because the remote procedure call (RPC) runtime services authenticate the client when the connection is requested. In much the same way as with the Named Pipes Net-Library, the Multiprotocol Net-Library requires a valid Windows NT account.

Enabling the Windows NT **guest** account is one way of dealing with users who do not have a Windows NT account, but who want to connect using the Named Pipes or Multiprotocol Net-Libraries. When these users request a session, they can connect to Windows NT as the **guest** user account, and then attempt to log in to SQL Server. Because enabling the **guest** account makes your entire Windows NT environment less secure, this option is not usually recommended, and is only mentioned here as a

Upgrading from SQL Server 6.5

work-around of last resort.

The security model for SQL Server 6.5 changed from SQL Server 6.0, and has changed again in 7.0. The basic approaches of the prior versions were discussed in “Permissions System.” These changes were necessary to provide a practical working environment for SQL Server. Because of this change, permissions need to be carefully considered when performing an upgrade.

Upgrade Considerations

The information covered in this section applies only to upgrading from SQL Server 6.5 Integrated Mode or Mixed Mode. If the upgrade is performed from a computer running SQL Server 6.5 that is configured in standard mode, no security problems will be encountered. However, Microsoft recommends that the new security functionality available as part of Windows NT Authentication Mode be used in the upgraded environment.

Upgrade Process

The upgrade process can be performed on one computer, or remotely from one computer to another. Logically, a single computer upgrade is the same as a two-computer upgrade where the source and target computers are the same. The two computers will be referred to as source and target servers. The source server is expected to have SQL Server 6.0 or SQL Server 6.5 installed, and the target server is required to have SQL Server 7.0 installed.

During the version upgrade process, a program is executed that opens the SQL Server 6.5 integrated registry key on the source computer, and reads the SIDs of all accounts that have integrated logins granted to them. The accounts that have integrated security configured on the source server may be Windows NT global groups, Windows NT local groups, or Windows NT users. In the case of the global groups and users, these can be from a local domain (if SQL Server 7.0 is running on a member server, or they could be accounts from a trusted domain). If SQL Server 7.0 is installed on a domain controller, the local groups would be the local groups from the domain controller's domain; otherwise, the local groups would be the local groups of the member server.

Accounts that were given administrative privileges on the source server are ignored in the drill-down and account-mapping processes.

Note: The **sp_grantlogin** statement will be executed on the computer running SQL Server 7.0 for each Windows NT account that was configured to use integrated security on SQL Server 6.5. The SQL Security Manager that shipped with SQL Server 6.5 actually executed **xp_grantlogin**, so the upgrade process is mimicking what was done in the previous version.

Analyzing the Upgrade Output

Most of the security-related upgrade difficulties are based on the fact that in SQL Server 6.5 integrated security was implemented by securing a key in the registry, and only those who had access to that key could gain access to the server. For more information, see "Use of SIDs." The permissions on the registry key were linked to user login accounts, which were stored in the **syslogins** table.

SQL Server 7.0 no longer relies on this method for securing access to the server. Instead, it allows granting of access to the server based on Windows NT user or group SIDs. Therefore, the upgrade process is

sometimes unable to identify what the original security requirements were. This is usually because the SQL Server security environment was not up-to-date, or because the upgrade is going into a different environment.

The following table has been compiled to explain how logins appear in SQL Server Enterprise Manager after an upgrade.

Name	Type	Server Access	Default DB	User	Line
user1	Standard	Permit	**master**		1
a#user2	Standard	Permit	**master**		2
BUILTIN\ Administrators	Windows NT Group	Permit	**master**	**dbo**	3
DOM3\SQLUsers	Windows NT Group	Permit	**master**		4
DOM3_user#3	Standard	Permit	**master**		5
DOM3_Administrator	Standard	Permit	**master**		6
REDMOND\a user4	Windows NT User	Via Group	**master**		7
REDMOND\user5	Windows NT User	Via Group	**master**		8

The contents of this table are explained in the sections that follow.

User Has Been Deleted

Lines 1 and 2 in the previous table are produced when the users are not found in the Windows NT User Directory. Specifically, if the **xp_logininfo** system stored procedure does not return the username, it is converted as a standard login as in these two lines. The '#' character in line 2 is used to represent the space, as SQL Server 6.5 and earlier did not support special characters.

Administrator Account

The *BUILTIN\Administrators* local group on line 3 of the previous table has been aliased to the **dbo** user of the **master** database.

Users of a Trusted Domain

Line 4 in the table refers to the **DOM3\SQLUsers** group, a global group on a trusted domain. The members of this group have been granted login rights. However, the members of this group have also been granted login rights using standard security using their username as they would have

appeared in SQL Server 6.5 and earlier. This has been done to provide backward compatibility for standard mode security.

Notice the entry for the **Administrator** account on line 6. The Windows NT Administrator account of the **DOM3** domain was granted user login rights prior to the upgrade. This has been preserved. All user level logon access will be processed in this way.

Users of the Current Default Domain

Users of the current default domain (as configured in SQL Server 6.*x* prior to the upgrade) are upgraded as per lines 7 and 8. Notice the type of the account and the existence of the space in line7. SQL Server 7.0 now supports special character in the account names.

How to Reduce the Upgrade Risk

The best method of insuring that the security settings of SQL Server 6.x are upgraded in the most constructive way possible, is to plan the upgrade thoroughly and prepare the security environment.

Preparing SQL Server 6.5 Security Environment

Microsoft strongly recommends that all security settings be cleaned up thoroughly prior to the upgrade. SQL Security Manager should be run to ensure that all Windows NT accounts are in sync with SQL Server. If the environment is in order, the upgrade process has the highest possible opportunities to complete properly.

Step Through the Upgrade

It is relatively easy to monitor the upgrade process and identify how the upgrading of user accounts and groups is likely to go. The SQL Server Upgrade Wizard allows the process to be stopped after every step. If the option to stop after each step is selected, the user can analyze the output created by the early stages of the security upgrade. The specific files to analyze are **loginmap.sid** and **loginmap.txt**. If the content does not appear correctly, these text files can be edited prior to continuing. Use care when editing the **loginmap.sid** and **loginmap.txt** files. Microsoft does not support the editing of these files during the upgrade process However, take great care when doing this, as Microsoft does not support the editing of these files during the upgrade process.

Do Not Upgrade to a New Domain

When upgrading from SQL Server 6.*x* using the tape method, do not back up the database in one domain and then upgrade it into another domain. When the **xp_logininfo** stored procedure is executed, it will probably not find any of the accounts that existed in the original domain (and if it does, they are probably not the correct accounts, but are just identical in name). The login rights will be treated as if the accounts have been deleted. For more information, see “User Has Been Deleted.”

Character Mapping Not Required

In SQL Server 6.5 and earlier, character mappings had to be configured to deal with Windows NT account names, which contained special characters such as the backslash (\) character. Therefore, SQL Server 6.5 and earlier provided three mapping characters: “#”, “_” and “$”. These character mappings are no longer required, as SQL Server 7.0 can deal with spaces and backslashes in account names.

Do Not Use sa Account

In SQL Server 6.5 and earlier, administrators logged in to SQL Server using the system administrators (**sa**) account to perform most administrative tasks. This often required a large number of people to have administrative access.

All Windows NT users who are given **sa**-type rights on SQL Server 7.0 should be assigned to the **sysadmin** fixed server role.

For more information, see “Do Not Use the sa Account.”

Do Not Use Aliasing

SQL Server 7.0 supports aliasing of user accounts within a database for backward compatibility, and its implementation is no longer recommended. Microsoft now recommends the use of roles. Roles are more powerful, and they provide similar functionality to aliasing. For more information, see “User-Defined Roles.”

Setting Up a Secure SQL Server 7.0 Installation

The information discussed in this section applies to SQL Server 7.0 installed on Windows NT only, as the Windows 95 and Windows 98 environments do not provide the security features discussed.

This section assumes that SQL Server 7.0 has been configured with Windows NT Authentication Mode to provide the highest level of security.

Do Not Use the sa Account

Microsoft recommends that all administrators of SQL Server be granted access to SQL Server through Windows NT group membership, and that this same group be made a member of the **sysadmin** server role. This approach has one minor drawback. Windows NT administrators can give anyone **sysadmin** privileges on SQL Server 7.0, as they are able to add any user to the appropriate Windows NT group.

If a site does not want to give Windows NT administrators the ability to give others (or themselves) **sysadmin** access to SQL Server, only individual Windows NT accounts should be assigned to the role of **sysadmin**.

In each case, it is strongly recommended that the **sa** account not be used for day-to-day administration, but rather, that a password be assigned; the password should be hard to break, and then be locked in a safe for emergency access only.

If you are running SQL Server 7.0 with Windows NT Authentication Mode (as recommended in this document) you cannot log on using the **sa** account, as only trusted connections are allowed.

Note: Even thought the **sa** account can not be used to log in to SQL Server 7.0 when it is running in Authentication Mode, it is still important to assign an **sa** password. This is because a small change in the registry can change the security mode from Authentication Mode to Mixed Mode.[19] If the **sa** password is blank (as per a default installation), an intruder (or the Windows NT Administrator) would be able to gain access to the server. For information on ways to reduce the chance of such an attack, see “Registry.”

[19] The registry key where the login mode is configured is:

HKLM/Software/microsoft/MSSQLServer/MSSQLServer/
LoginMode

If the value is 0 then Mixed Mode has been configured, if it is a 1 then Windows NT Authentication Mode has been selected.

Service Accounts

SQL Server 7.0 runs as three Windows NT services: MSSQLServer, SQLServerAgent, and Microsoft Search. The MSSQLServer service is the engine that provides the core functionality of SQL Server. The SQLServerAgent service provides the capability to schedule regular commands, schedule replication, provide a method for dealing with errors, contact SQL Server operators when errors occur, as well as other support functions. The Microsoft Search service provides the full-text search capability, and must always be configured to use the local system account. The MSSQLServer and SQLServerAgent services can be configured to use one of the following types of Windows NT accounts:

- Local service account
- Local user account
- Domain user account

The selection depends on the functionality that is required for SQL Server 7.0. Both services can be configured to use the same Windows NT user account.

If the service account needs to be changed after the server has been installed, the SQL Server Enterprise Manager should be used. While it is also possible to change the service account for the MSSQLServer and SQLServerAgent services in Control Panel, this is not recommended because the configuration details for the Microsoft Search service are not kept in sync.

The changes to account information take effect the next time the service is started. The MSSQLServer and SQLServerAgent services can be configured to use different Windows NT user accounts, although this is not usually recommended. When changing the service account, the changes must be made to both services, as they are configured separately.

One consideration that can reduce administrative overhead in a multi-server environment is the use of one domain user account for all SQL Server 7.0 servers in the enterprise.

Local System Account

SQL Server 7.0 can be run using the local system account if the SQL Server is not configured for replication and does not require access to network resources.

The following permissions must be set for the local system account in order for SQL Server 7.0 to perform its tasks properly:

- Full Control on the SQL Server directory (by default \MSSQL7)
- Full Control on all .mdf, .ndf, and .ldf database files
- Full Control on the registry keys at and under:

- HKEY_LOCAL_MACHINE\Software\Microsoft\MSSQLServer
- HKEY_LOCAL_MACHINE\System\CurrentControlset\Services\MSSQLServer

Local User Account

If SQL Server 7.0 is configured to use a Windows NT local user account, the same restrictions apply as for local system, with the following addition:

- The user account must be granted the **Log On As A Service** right.

Domain User Account

Configuring SQL Server 7.0 with a domain user account provides the greatest level of flexibility. Some examples of functionality available only when a domain user account is used, include:

- Replication
- Backing up to and restoring from network drives
- Performing heterogeneous joins that involve remote data sources
- SQL Server Agent mail features and SQL Mail

In order for SQL Server 7.0 to perform its tasks, the domain user account must be configured as the local user account discussed earlier. However, some extended functionality is available only if further permissions are considered. This is best outlined in the following table.

Service	Permission	Functionality
MSSQLServer	Network write permissions.	Ability to read/write to remote backups, data loads, and so on.
MSSQLServer	Act as part of the operating system and replace process level token.	Run **xp_cmdshell** for a user other than a SQL Server administrator.
SQLServerAgent	Member of the **Administrators** local group.	Create **CmdExec** and **ActiveScript** jobs belonging to someone other than a SQL Server administrator.
SQLServerAgent	Member of the **Administrators** local group.	Use the autorestart feature.
SQLServerAgent	Member of the **Administrators** local group.	User run-when-idle jobs.

In order to provide maximum functionality to SQL Server 7.0, it is recommended that the domain user account be a member of the **Administrators** local group.

File System

Windows NT provides an excellent security framework for securing operating system objects such as files. Microsoft recommends that NTFS file permission be applied to the data and log files of all databases. The user account that SQL Server 7.0 is configured to use must be given Full Control permissions on the database files.

Further, all SQL Server 7.0 files, including executables and dynamic link libraries (DLLs), should be configured so that users cannot manipulate them. Permissions on these files should be set to allow the user account that SQL Server uses, the **Administrators** group and local system accounts Full Control permissions. No other permissions should be set.

Registry

In order to secure the SQL Server 7.0 installation from security attacks by users who have logon rights on the physical server, it is prudent to set Windows NT permissions on the registry keys that are used to configure SQL Server 7.0.

Specifically, all the keys under HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\MICROSOFT SQL SERVER 7.0\ should be secured. The **everyone** group permissions on this key should be removed, and Full Control permissions added for the **Administrators** group, the local system account or the SQL Server service account.

Setting permissions on the registry keys is particularly important if the SQL Server administrators want to stop the Windows NT administrators from accessing the SQL Server. In this case, the SQL Server administrators should also take ownership of the registry key, and remove permissions from the **Administrators** group. It is then imperative that the SQL Server service account has Full Control permissions. Although this does not stop administrators from gaining access, it allows SQL Server administrators to know when the Windows NT administrators have compromised security. Administrators can always take ownership, but they cannot give it. For more information regarding Windows NT administrators gaining access to SQL Server, see “Do Not Use the sa Account.”

Auditing

When SQL Server 7.0 is using Windows NT Authentication Mode, it provides the capability to audit logons to the server. The audit level can be configured using SQL Server Enterprise Manager, or by using the **sp_loginconfig** stored procedure.

Possible auditing settings are:

- **None** logs no auditing information
- **Success** causes only successful logins to be logged

- **Failure** causes only failed logins to be logged
- **All** causes successful and failed logins to be logged

The auditing information is written to the SQL Server 7.0 error log.

Profiling for Auditing

SQL Server 7.0 provides a very powerful profiler, which allows the analysis of many internal events that occur within SQL Server.

SQL Server Profiler works by capturing all the actions performed on the SQL Server, and sending them to the SQL Server Profiler, where they can be analyzed. The capture can be viewed real-time on the screen, saved to a text file, or inserted into a SQL Server table.

The SQL Server Profiler allows capturing of virtually all events that take place within SQL Server, including:

- Login Failed
- Locking: Deadlock
- Object: Closed
- Stored Procedure: Statement Starting
- Session: Disconnect
- RPC: Completed

This information can provide excellent support to establish who did what and when.

Backup and Restore

Security of Backup Files and Media

The most secure method for backups is to use SQL Server 7.0 to back up to data files, and then to use the Windows NT backup program to back the data files up to backup media using the password feature. This ensures that only those who know the password can restore the files.

The backup data files should be on an NTFS partition with directory permissions set to prevent the ordinary user from gaining access to the files.

If backup media can be physically secured, the standard SQL Server 7.0 backups will not pose any security risks.

Restoring to Another Server

Three specific situations will be addressed with regard to restoring the database to another server. The first situation applies where the old server (where the database originated from) and the new server (where the database is going) are using Mixed Mode for authentication. The second and third situations apply where Windows NT Authentication Mode is used; the difference between these is that the second situation addresses where a database is restored to a server in the same domain, while the third scenario applies to a database being restored to a server in another domain.

Other scenarios could be discussed here, such as restoring a database from a server using Mixed Mode to a server using Windows NT Authentication Mode and vice versa. These scenarios have not been included here, as they are very similar to those outlined in the following sections.

Mixed Mode

When restoring a database to a server using Mixed Mode for security authentication, the database security breaks. This is because the logons are maintained in the **sysxlogins** table in the **master** database, while the user's rights to access a database are stored in the **sysusers** table of the respective database; a logical link is maintained between the user's entry in the **sysxlogins** table and the user's entry in the **sysusers** table. This link is a generated 16-byte GUID. For more information, see "Generation of GUIDs for Non-Trusted Users."

The net effect of the GUID implementation for Mixed Mode authentication is that when a database is restored to a computer running SQL Server 7.0, other than the one where the database access was granted, the link between the **sysxlogins** table and the **sysusers** table breaks, thereby effectively granting access to the database to no one. Members of the **sysadmin** group are an exception to this. All role memberships and user permissions would have to be re-created.

Restoring a database to another server shows one of the weaknesses of Mixed Mode authentication.

Windows NT Authentication (Same Domain)

If the database is restored to another computer running SQL Server 7.0 in the same domain, the permissions in the database remain intact. The only consideration here is whether users are granted permission to log on to the server. The permission to log on to the server is implemented at each instance of SQL Server.

For example, Bob is a member of the **SALES** group, and the **SALES** group is granted login permissions at SQLSERVER1. Bob is granted database access rights to the sales database. When the sales database is restored to SQLSERVER2, Bob's permissions still exist in the **sales** database. However, because the **SALES** group is not granted login rights to the server, Bob cannot use the database. If the administrator grants the **EVERYONE** group login rights to the server, Bob can use the database. This is because the only restriction stopping Bob from using the **sales** database was logging in to the server.

When restoring a database to another server in the same domain, the permissions within the database remain intact, but the permissions to log in to this specific server may need to be granted.

Windows NT Authentication (Different Domain)

When restoring a database to another domain, a number of scenarios should be considered. These scenarios apply to users who want to access the database.

Users from a Trusted Domain

If a Windows NT trust relationship has been established between the old and the new domain, such that the new domain trusts the old domain, the users from the old domain may use the database with all permissions intact, provided that they have been granted the right to log in to SQL Server.

Users from other trusted domains would not have rights to access the database, much like the users from the new domain.

Users from the New Domain

None of the users form the new domain will have access, as their SIDs do not exist in the **sysusers** table of the database.

The only exception to this are the **BUILTIN** accounts of Windows NT. As these accounts always have the same SIDs on all servers, any permissions granted to a **BUILTIN** account, such as the local **Administrators** group, remain intact. This assumes that the **BUILTIN** accounts have logon rights, and that SQL Server is installed on a domain controller.

Users from Any Domain with Same Username and Password

In most Windows NT security implementations, when access is required to a resource that is not in the user's own domain, the user is able to access the resource providing that a user account exists with the same username/password combination. This behavior is transparent.

Provided that the user is using named pipes to connect to the server, this will work if the user establishes a connection to a file share first. This method also works if a user wants to use an account of another name, providing that he/she is running Windows NT as the computer operating system. If a user is denied access when connecting to a file resource from a computer running the Windows NT operating system (and the user is not currently using any other credentials on the computer being connected to), the opportunity is given to provide a username and password for logon purposes.

Attaching/Detaching Database Files

The issues associated with attaching and detaching database files are identical to those discussed in "Restoring to Another Server." An exception is the requirement to create the database before restoring the data.

General Windows NT Security Configurations

It does not take much thought before one realizes that one could write one hundred pages or more on Windows NT security. As SQL Server 7.0 relies on the Windows NT security architecture, all security principles that apply to Windows NT also apply in some way to Windows NT-based servers running SQL Server 7.0.

In an attempt to have a more focused document, only security issues that apply to SQL Server 7.0 specifically have been addressed to this point. However, there are a couple of other items that should be mentioned briefly.

Disable Windows NT Guest Account

When running SQL Server 7.0 in Windows NT Authentication Mode, the server relies on Windows NT to perform all authentication of clients. This brings with it the security framework that applies to Windows NT, both the strengths and the weaknesses. Fortunately, there are not many of the latter. However, one issue that has been adequately documented in many Microsoft and third-party security papers is the use of the Windows NT **Guest** account. It is strongly recommended that the **Guest** account be disabled, if this has not already been done.

For more information, see the TechNet resource listed in “Web Sites.”

Restrict Physical Access

As with any Windows NT-based server, it is recommended to restrict physical access wherever possible. One of the risks of unauthorized physical access is the ability by an intruder to start the server from a floppy disk and gain access to the Windows NT file system. Your mission-

Conclusion

critical production database servers should be physically secured.

With the release of SQL Server 7.0, database administrators have the power to configure and manage secure database servers that tightly integrate with Windows NT security.

Programmers have also been provided with the means to develop secure applications without increasing the development overhead.

For Microsoft, this release could not have come at a better time; a time when the uptake of databases is exploding in the Internet commerce space.

For the readers of this document, there is no better time to implement a

Appendix A: Finding More Information

strategic deployment of SQL Server 7.0 than now.

Books

Sams Teach Yourself Microsoft SQL Server 7.0 in 21 Days by Richard Waymire and Rick Sawtell. Copyright 1999, Sams Publishing. ISBN 0-672-31290-5.

Inside Microsoft SQL Server 7.0 by Ron Soukup and Kalen Delaney. Copyright 1999, Microsoft Press. ISBN 0-735-60517-3.

Web Sites

http://www.microsoft.com/sql –SQL Server home page

http://support.microsoft.com/support/sql –SQL Server support home pages

http://technet.microsoft.com –TechNet resource site

http://msdn.microsoft.com –MSDN resource site

http://www.microsoft.com/security – Microsoft's central site for security information

White Papers

http://www.microsoft.com/sql/index.htm#W – Contains a complete list of all white papers available on the SQL Server Web site.

Appendix B: Security Related Stored Procedures

sp_addalias
sp_addapprole
sp_addgroup
sp_addlinkedsrvlogin
sp_addlogin
sp_addremotelogin
sp_addrole
sp_addrolemember
sp_addserver
sp_addsrvrolemember
sp_adduser
sp_approlepassword
sp_change_users_login
sp_changedbowner
sp_changegroup
sp_changeobjectowner
sp_dbfixedrolepermission
sp_defaultdb
sp_defaultlanguage
sp_denylogin
sp_dropalias
sp_dropapprole
sp_dropgroup
sp_droplinkedsrvlogin
sp_droplogin
sp_dropremotelogin
sp_droprole
sp_droprolemember
sp_dropserver
sp_dropsrvrolemember
sp_dropuser
sp_grantdbaccess
sp_grantlogin
sp_helpdbfixedrole
sp_helpgroup
sp_helplinkedsrvlogin
sp_helplogins
sp_helpntgroup
sp_helpremotelogin
sp_helprole
sp_helprolemember
sp_helprotect
sp_helpsrvrole
sp_helpsrvrolemember
sp_helpuser
sp_password
sp_remoteoption
sp_revokedbaccess
sp_revokelogin
sp_setapprole
sp_srvrolepermission
sp_validatelogins

Appendix C: Changes to SQL-DMO Object Model

This appendix was initially intended to refer to security related objects only, but as this information is of general interest to all developers, the complete changes have been included here.

Removed SQL-DMO Objects/Collections

The following objects were removed:

- Tasks
- Task

Changed SQL-DMO Objects/Collections

The following objects were renamed:

- Devices to BackupDevices
- Device to BackupDevice
- Executive to JobServer
- HistoryFilter to JobHistoryFilter

The following object has been set aside for special use only[20]:

- Group

[20] This object is no longer available for developers to use. It remains here because the upgrade process uses it.

New SQL-DMO Objects/Collections

The following objects were added.

- FileGroups
- FileGroup
- DBFiles
- DBFile
- LogFiles
- LogFile
- ServerRoles
- ServerRole
- DatabaseRoles
- DatabaseRole
- Restore
- LinkedServers
- LinkedServer
- LinkedServerLogins
- LinkedServerLogin
- JobFilter
- JobCategories
- JobSchedules
- JobSchedule
- Jobs
- Job
- JobSteps
- JobStep
- TargetServers
- TargetServer
- TargetServerGroups
- TargetServerGroup
- AlertCategories
- OperatorCategories
- Category
- Schedule